Personalizzare TEXworks

Filippo Vomiero

Sommario TEXworks è l'editor incluso nelle installazioni di TEX Live per Windows e MiKTEX: a prima vista può sembrare molto semplice, soprattutto se comparato con i più complessi TeXstudio o TeXmaker. Tuttavia, TEXworks è anche estremamente personalizzabile e questo lo rende versatile e facilmente adattabile alle proprie esigenze. Questo articolo illustra come personalizzare alcuni aspetti di TEXworks: il tema dell'interfaccia, la colorazione della sintassi, i dizionari e gli strumenti di composizione.

Abstract TEXworks is the default editor bundled with TEX Live for Windows and MiKTEX: at first glance it seems a quite minimalist program, even more so if compared to the more feature-rich TeXstudio or TeXmaker. TEXworks, however, is also highly customizable, so it is more suitable to be crafted fitting one's own needs. The article will show a few methods to customize TEXworks features like interface theme, syntax highlighting, dictionaries and processing tools.

1. Introduzione

Per alcuni anni, come passatempo, mi sono divertito a decompilare applicazioni Android per modificare la combinazione di colori predefinita, soprattutto a partire da Android 5 con l'introduzione del *Material design* che ha reso tutte le applicazioni a sfondo bianco, scelta che ho sempre trovato molto fastidiosa per gli occhi, soprattutto se si legge di sera con poca luce. Passando al mondo LATEX, è stato per me naturale provare a modificare anche TEXworks, il programma incluso nelle installazioni di TEX Live per Windows e MiKTEX.[1]

TEXworks è un programma open source, multi-piattaforma, sviluppato con la libreria Qt, ispirato a TeXShop, l'attuale editor predefinito in MacTeX. Nella mia ricerca ho constatato che il programma è ampiamente configurabile, sia per quanto riguarda le funzionalità, sia nell'estetica dell'interfaccia: l'articolo mostrerà sul piano delle funzioni come aggiungere e configurare dizionari e strumenti di composizione, mentre sul piano dell'estetica come modificare il tema dell'interfaccia e la colorazione della sintassi.

Prima di continuare, una premessa importante: TEXworks è multi-piattaforma e viene rilasciato per Windows, per le principali distribuzioni Linux e per macOS[2]; nell'articolo, tuttavia, tutti gli esempi si riferiscono a Windows. Nella maggior parte dei casi il cambio di sistema operativo non dovrebbe costituire un problema, visto che sono coinvolti linguaggi svincolati dalla piattaforma (*regex* e *Qt*), ma molto probabilmente le parti che riguardano i file di configurazione sono valide solo in Windows, e necessitano di qualche adattamento per essere applicate negli altri sistemi operativi. La questione va oltre le mie capacità, ma la

1. Esistono programmi molto validi come TeXstudio e TeXmaker che supportano nativamente i temi e danno la possibilità di impostare un tema scuro. TEXworks, tuttavia, è un programma ben più leggero e rapido nell'esecuzione, oltre a garantire una maggior libertà di configurazione.
2. In GitHub sono disponibili anche i sorgenti che possono essere compilati per altre piattaforme.

comunità del GUIT è ricca di esperti in grado di risolvere questi piccoli problemi, e il forum esiste anche per questo.

2. La cartella delle risorse

Il primo passo da compiere per personalizzare TeXworks è capire dove sono memorizzati i file di configurazione. Nel menu `aiuto` di TeXworks, è presente la voce 'preferenze e risorse', che permette di individuare dove sono memorizzate tali informazioni. Le preferenze sono salvate, di norma, in un file `texworks.ini`, o, se stiamo lavorando in Windows, possono essere salvate nel registro di configurazione di sistema. Le preferenze si possono modificare direttamente da GUI, e vengono salvate in automatico. Le risorse invece sono memorizzate in una cartella dedicata che contiene vari file di testo corrispondenti a specifiche funzioni: per personalizzare l'installazione di TeXworks vanno modificati manualmente, e nelle prossime sezioni vedremo come fare.

La cartella delle risorse contiene al suo interno queste sotto-cartelle:

completion Contiene i file con tutte le voci per la funzione di auto-completamento

configuration Cartella con file di configurazione per varie funzioni:

- rientro automatico
- coppie di delimitatori
- virgolette intelligenti
- colorazione della sintassi
- *tag* di sezione (visibili dal menu Finestra, visualizza, tags)
- configurazione avanzata, per delle impostazioni non modificabili dal menu preferenze

dictionaries Contenitore per i dizionari disponibili per il controllo ortografico

scripts TeXworks permette di eseguire degli script per svolgere funzioni aggiuntive, nella cartella sono presenti alcuni esempi

translations Cartella per testare localmente una traduzione dell'interfaccia

TUG Posizione di default del file `texworks.ini`

In ambiente Windows e con TeX Live, di default viene creata una cartella, il cui nome corrisponde alla versione di TeX Live in uso, dentro alla cartella principale dell'account utente, ad esempio:

`C:\Users\`⟨*nome utente*⟩`\.texlive2021`

Le preferenze e le risorse di TeXworks vengono salvate in un'ulteriore sotto-cartella `configuration`. Ad ogni cambio di versione si rende quindi necessario trasferire tutti i file personalizzati nella cartella della nuova versione (ad esempio `.texlive2022`). Ci sono due possibili strade per risolvere il problema. La prima consiste nel creare un file `texworks-setup.ini` nella cartella con l'eseguibile di TeXworks.[3] All'interno del file vanno indicati i percorsi da utilizzare per salvare i file di configurazione:

3. In Windows va usata la cartella:
⟨*TeX Live*⟩`\tlpkg\texworks\texworks.exe`

```
inipath=C:/myfolder/TW_conf/
libpath=C:/myfolder/TW_conf/
```

La prima voce, `inipath`, indica la posizione del file `texworks.ini`, mentre `libpath` indica la posizione della cartella delle risorse. L'esempio qui riportato, ripreso dal manuale TEXworks (Delmotte *et al.*, 2021, 29–30), usa `TW_conf`; ciascuno può scegliere il nome che preferisce, ma la cartella non verrà creata in automatico. Si noti, inoltre, l'uso della barra dritta al posto di quella rovescia solitamente usata in ambiente Windows. Se si usa TEX Live, può essere una scelta efficace usare la cartella personale `texmf` prevista dalla *TEX Directory Structure* (TDS).[4]

La seconda soluzione, valida solo in ambiente Windows, consiste nel modificare il collegamento che si usa per aprire TEXworks e usare esclusivamente quello (può apparire scomodo, ma è necessario anche per usare un tema personalizzato, come si vedrà nella sezione 5). Il collegamento all'eseguibile di TEXworks creato durante l'installazione di TEX Live punta a:

⟨*TEX Live*⟩`\bin\win32\texworks.exe`

Se si modifica in:

⟨*TEX Live*⟩`\tlpkg\texworks\texworks.exe`

le impostazioni verranno salvate nel registro di configurazione, mentre la cartella delle risorse verrà salvata in:

`C:\Users\`⟨*utente*⟩`\AppData\Roaming\TUG\TeXworks`

risultando così immune ai vari cambi di versione annuali.

3. I dizionari

Una volta individuata la posizione della cartella di configurazione del programma, possiamo iniziare con le personalizzazioni. La più semplice è quella di aggiungere la cartella con i file dei dizionari, così da poter usufruire del controllo ortografico durante la digitazione (cfr. Gregorio, 2010, 12–13). Il controllo è effettuato dal programma Hunspell, lo stesso usato da Open/Libre Office, Firefox e Thunderbird. Per usufruire di un dizionario è sufficiente creare la cartella `dictionaries` dentro alla cartella delle risorse; al suo interno inseriremo i dizionari delle lingue in cui siamo soliti scrivere.[5] È possibile recuperare i file dei dizionari dai repository di estensioni di OpenOffice o LibreOffice:
`https://extensions.openoffice.org/`
`https://extensions.libreoffice.org/`

Una volta fatto, entriamo nella finestra 'preferenze' di TEXworks (menu `modifica`, preferenze), selezioniamo la scheda 'editor' e nel menu a tendina corrispondente alla voce 'lingua verifica ortografia' potremo selezionare la lingua di uno dei dizionari installati.

TEXworks viene anche distribuito con uno script automatico in grado di rilevare la lingua di un documento indicata nelle opzioni di babel e impostare la verifica dell'ortografia di

4. L'esatta posizione di `texmf` cambia a seconda del sistema operativo: in Windows si trova in `%USERPROFILE%\texmf`, in macOS e nei sistemi Linux di norma è in `$HOME/texmf`.
5. La procedura è valida per Windows e macOS; nei sistemi GNU/Linux invece i dizionari sono in una cartella condivisa da tutti i programmi che ne fanno uso; per questo motivo è possibile che i dizionari richiesti siano già installati. Il percorso è `/usr/share/myspell/dicts`.

conseguenza. Lo script, tuttavia, non è configurato per la lingua italiana, per cui è necessaria una piccola modifica. Il file dello script si chiama `babelLanguage.js` (cartella `scripts/Hooks`), al cui interno va aggiunta la seguente stringa alla lista di lingue supportate:

```
spellingDict.italian = "it_IT";
```

4. Strumenti di composizione

Un altro tipo di personalizzazione sul piano delle funzionalità riguarda la possibilità di aggiungere o modificare gli strumenti di composizione disponibili in TeXworks e accessibili tramite il menu a tendina in alto a sinistra nell'interfaccia. I principali strumenti di composizione sono già supportati nativamente, ma nella finestra preferenze, alla scheda composizione, è possibile aggiungere, rimuovere o modificare gli strumenti, oltre a cambiare l'ordine in cui appaiono nel menu a tendina e stabilire lo strumento di default.

Nella mia installazione ho aggiunto, per i tre motori principali (pdfLaTeX, LuaLaTeX e XeLaTeX), una versione alternativa con l'argomento `-shell-escape`, che permette al programma di lanciare programmi esterni, come gregorio o abcm2ps per la scrittura di musica con LaTeX. La procedura è molto semplice: dalla scheda composizione nelle preferenze, si preme il pulsante "+" per creare un nuovo strumento e si procede con la compilazione dei campi. Vediamo un esempio con pdfLaTeX:

```
Nome: pdfLaTeX+se
Programma: pdflatex.exe
```

Gli argomenti si aggiungono usando il pulsante "+" sulla destra, ne servono tre:

```
--shell-escape
$synctexoption
$fullname
```

Il primo argomento è ovvio, il secondo serve a passare al programma di compilazione l'opzione `-synctex=1` per permettere la sincronizzazione tra il file sorgente e il `pdf`. Il terzo argomento passa al programma il nome completo di estensione del file da cui avviamo la compilazione.

Ulteriori informazioni si trovano sulla pagina GitHub di TeXworks:

```
https://github.com/TeXworks/texworks/wiki/AdvancedTypesettingTools
```

4.1. Configurazioni aggiuntive

Se avete impostato TeXworks per utilizzare anche altri strumenti di composizione, potreste desiderare che i file prodotti da quegli strumenti (con le relative estensioni) vengano riconosciuti dal programma nella finestra di apertura dei file. Per ottenere questo, bisogna modificare il file `texworks-config.txt` che si trova nella cartella `configuration`. Una volta aperto il file, verso la fine si vedono una serie di righe commentate che iniziano con `# file-open-filter:`. Le righe vanno de-commentate per abilitare la personalizzazione dei tipi di file supportati (come spiegato nel file stesso). A quel punto, si potranno inserire nuove righe con le definizioni dei file che vogliamo implementare.[6] Ad esempio, per i file usati da GregorioTeX, va aggiunta

6. Esula dallo scopo di questa guida approfondire, ma va segnalato che il programma è così versatile che inserendo la definizione relativa a un tipo specifico di file e costruendo un relativo set di regole per la colorazione della sintassi

una riga in questo modo:

```
file-open-filter: Gabc score (*.gabc)
```

Sempre nello stesso file è possibile anche impostare degli schemi per la pulizia dei file ausiliari: per i programmi principali ci sono già le definizioni necessarie, ma se gli strumenti di composizione aggiunti producono a loro volta dei file ausiliari con estensione specifica, è possibile includerli negli schemi per la pulizia. L'operazione è spiegata bene, con tanto di esempio, nella pagina di documentazione di GregorioTeX relativa a TEXworks:

https://gregorio-project.github.io/configuration-texworks.html

5. Configurare un tema scuro

TEXworks non permette, per ora, di impostare un tema per l'interfaccia; tuttavia, il toolkit Qt con cui è compilato, permette di caricare un foglio di stile esterno con cui personalizzare l'aspetto dell'applicazione.[7]

Il primo passo è creare un file `css`, da salvare in una cartella dedicata; quando si lancia TEXworks va usata una opzione specifica per fargli caricare il foglio di stile (il percorso è quello mostrato nella sezione 2, valido in ambiente Windows):

"⟨*TEXworks*⟩`\texworks.exe" -stylesheet "`⟨*cartella foglio di stile*⟩`\`⟨*nome file*⟩`.css"`

Se digitato direttamente nel prompt dei comandi, non vanno usate le virgolette; la soluzione più pratica è quella di creare un collegamento aggiungendo alla chiamata del programma l'argomento appena visto.

5.1. Documentazione

La parte più impegnativa è costruire correttamente il foglio di stile; la soluzione illustrata nella sezione 5.2 serve a creare un tema a sfondo nero (visibile nella figura 1) ed è stata ottenuta dopo molti tentativi ed esperimenti. In rete non esiste molta documentazione specifica; io sono partito da una discussione su Stack Exchange (2017) generata dalla domanda "come posso ottenere un tema scuro su TEXworks?": la soluzione proposta da Robert Zhang mi ha dato lo spunto iniziale, e ho capito che con un po' di pazienza si poteva ottenere un buon risultato.

La seconda lettura obbligata è stata la ricca documentazione di Qt, che però va adattata caso per caso a ciò che è pertinente con l'interfaccia di TEXworks. Riporto qui di seguito le tre pagine più utili:

- https://doc.qt.io/qt-5/stylesheet-reference.html
- https://doc.qt.io/qt-5/stylesheet-examples.html
- https://doc.qt.io/qt-5/stylesheet-customizing.html

Da queste pagine si possono trovare ulteriori link per approfondire aspetti specifici.

(vedi la sezione 6), TEXworks idealmente può essere usato come editor per qualsiasi tipo di file, non necessariamente legati al mondo LATEX(come gli script in Javascript o i fogli di stile `css` per citare due tipi nominati in questo testo). Il sito sugli script di TEXworks mantenuto da Paul A. Norman illustra come configurare il programma per l'editing di file di script Qt: `http://twscript.paulanorman.info`

7. Al momento della scrittura di questo articolo, l'ultima versione di TEXworks è la `0.6.6`. Gli sviluppatori non escludono un futuro supporto nativo ai temi, ma non è possibile dire quando questo avverrà.

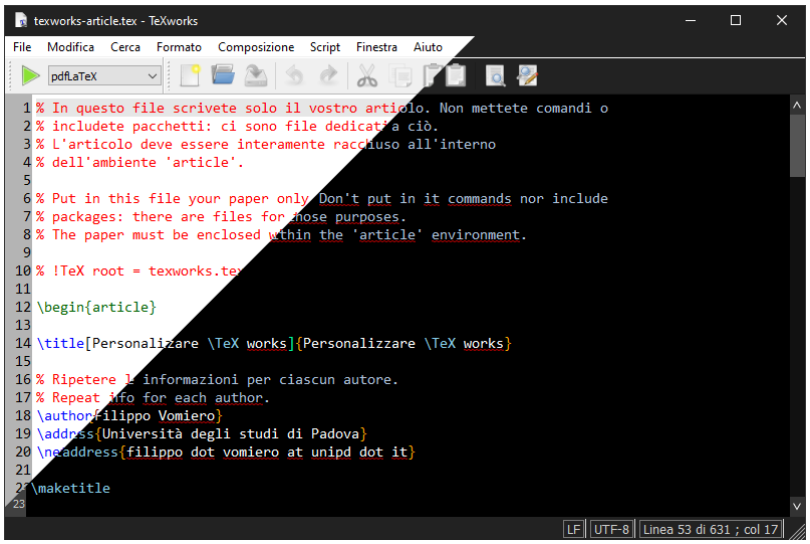

Figura 1. Tema chiaro e scuro a confronto.

5.2. Il foglio di stile

Un foglio di stile è un elenco di varie definizioni, così composte: la dichiarazione del nome dell'elemento che si vuole personalizzare e, tra graffe, le proprietà che si vogliono modificare (come, ad esempio, il colore di sfondo) con il valore da assegnare (nel caso di un colore il nome del colore o il suo valore `rgb`).

Ecco la prima definizione:

Definizione 1. QWidget

```
QWidget {
    background-color: #222222;
    selection-background-color: #555555;
    color: #cccccc;
    selection-color: white;
}
```

`QWidget` è l'elemento principale che stabilisce la combinazione di colori che verrà usata prevalentemente in tutta l'interfaccia di TeXworks. Le proprietà modificate sono il colore di sfondo e il colore in primo piano (usato per il testo). Si può notare come ci siano anche le definizioni per quando l'elemento è selezionato: queste proprietà relative allo stato sono importanti per ottenere un'interfaccia dinamica, che sembra rispondere alle azioni dell'utente.

Definizione 2. QTextEdit

```
QTextEdit {
    background-color: #000000;
    color: white;
    selection-background-color: #555555;
    selection-color: white;
}
```

Questa è la finestra dell'editor di testo. Le proprietà modificate sono le stesse di `QWidget`. In questo modo l'aspetto di TEXworks apparirà con una cornice grigio scuro, mentre la sezione principale dove si scrive sarà nera con testo bianco (oltre alla colorazione in base alla sintassi stabilita nella sezione precedente). Ho inserito anche uno sfondo diverso per il testo selezionato, come visibile nella figura 2.

Definizione 3. QComboBox

```
QComboBox {
    border: 1px solid gray;
    padding: 2px 18px 4px 6px;
    border-radius: 4px;
}
QComboBox::drop-down {
    subcontrol-origin: padding;
    subcontrol-position: top right;
    width: 20px;
    border-left-width: 1px;
    border-left-color: #eeeeee;
    border-left-style: solid;
}
QComboBox QAbstractItemView {
    border: 2px solid darkgray;
    selection-background-color: #333333;
    background-color: black;
}
QComboBox::down-arrow {
    image: url(./res/downarrow.png);
    width: 10px;
    height: 10px;
}
QComboBox::down-arrow:on {
    top: 1px;
    left: 1px;
}
```

`QComboBox` è l'elemento che riguarda i menu a tendina, come quello per scegliere il motore di compilazione in alto a sinistra nella finestra di TEXworks, visibile nella figura 3. La prima definizione riguarda la parte dove è presente il testo (per esempio pdfLaTeX), la seconda invece il riquadro della freccia. La terza definisce l'aspetto del menu una volta aperto, la quarta indica un'immagine per la freccia e le sue dimensioni, e la quinta la sposta a destra e in basso di un pixel (lo spostamento è ottenuto impostando un pixel di margine in alto e a sinistra) per dare l'impressione che il pulsante sia stato premuto.

Come si può notare, l'immagine per la freccia fa riferimento a una risorsa esterna, un'immagine png che va a rimpiazzare quella originale, la quale, essendo pensata per uno sfondo chiaro, è completamente nera e quindi poco visibile. L'immagine a cui si fa riferimento è una piccola freccia che ho disegnato con Inkscape e convertito in png. Conviene inserire tutte le immagini che andremo a utilizzare in un'unica cartella, nell'esempio io l'ho chiamata `res` (per *resources*), da posizionare nella stessa cartella dove abbiamo salvato il file `css`.

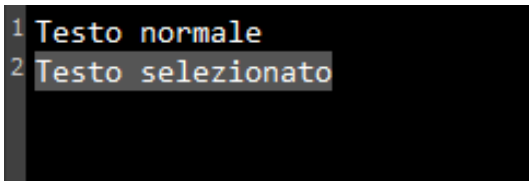

Figura 2. QTextEdit

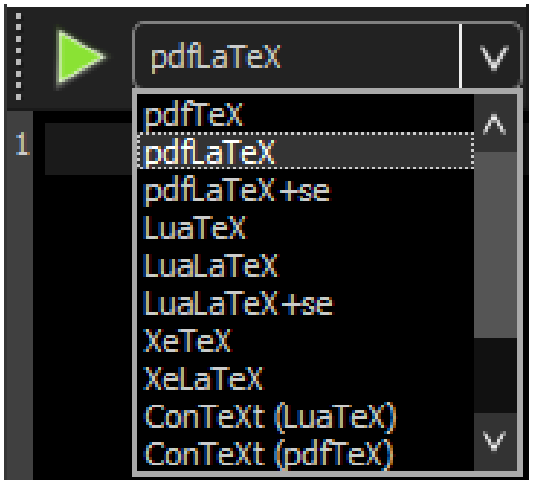

Figura 3. QComboBox

Definizione 4. QMenu

```
QMenu::item:selected {
    background-color: #333333;
}

QMenuBar::item:pressed, QMenuBar::item:selected {
    background: #333333;
}
```

Ho raggruppato qui due elementi che riguardano i menu principali: `QMenuBar` definisce l'aspetto dei pulsanti del menu principale (file, modifica, cerca, ecc.), mentre `QMenu::item` permette di definire l'aspetto delle voci di un menu una volta aperto. In entrambi i casi ho impostato un colore leggermente più chiaro quando una voce è selezionata (figura 4).

Definizione 5. QScrollBar:vertical

```
QScrollBar:vertical {
    background: #111111;
    width: 15px;
    margin: 20px 0 20px 0;
}
QScrollBar::handle:vertical {
    background: #555555;
    min-height: 20px;
}
QScrollBar::add-line:vertical {
     border: none;
     background: #333333;
     height: 20px;
     subcontrol-position: bottom;
     subcontrol-origin: margin;
}
```

```
QScrollBar::sub-line:vertical {
      border: none;
      background: #333333;
      height: 20px;
      subcontrol-position: top;
      subcontrol-origin: margin;
}
QScrollBar::add-line:vertical:pressed {
    background: #444444;
}
QScrollBar::sub-line:vertical:pressed {
    background: #444444;
}
QScrollBar::up-arrow:vertical {
    image: url(./res/uparrow.png);
    border: none;
    width: 8px;
    height: 8px;
    padding: 12px;
}
QScrollBar::down-arrow:vertical {
    image: url(./res/downarrow.png);
    border: none;
    width: 8px;
    height: 8px;
    padding: 12px;
}
QScrollBar::add-page:vertical, QScrollBar::sub-page:vertical {
   background: none;
}
```

`QScrollBar`, come dice il nome, riguarda le barre di scorrimento (figura 5), in questo caso quelle verticali, ma la definizione andrà replicata anche per quelle orizzontali. La prima definizione stabilisce il colore di sfondo della barra (più scuro rispetto al resto dell'interfaccia), la larghezza della barra e i margini: vengono lasciati 20 pixel di margine sopra e sotto, che è lo spazio che serve per disegnare i due pulsanti con le frecce. `Handle` è la parte in movimento della barra, qui disegnata più chiara; viene inoltre stabilita una grandezza (altezza) minima.

La terza e la quarta definizione creano due piccole barre sopra e sotto più chiare, che fungono da pulsante, e contengono le frecce. Vale la pena spiegare un paio di proprietà: `subcontrol-origin` indica da quale punto si stabilisce la posizione dell'elemento, in questo caso la linea del margine (che è il rettangolo più esterno), mentre `subcontrol-position` indica quale margine considerare, tra le quattro direzioni. Le due definizioni che seguono servono a rendere più chiaro il pulsante quando viene premuto.

È importante ricordare che quando si decide di modificare lo stile di un elemento, come in questo caso `QScrollBar`, la sintassi di Qt vuole che vengano definite tutte le sue parti: se non si indica un'opzione per gli elementi grafici, in questo caso le frecce, essi non verranno disegnati e al loro posto appariranno due piccoli quadrati. Si spiegano così le due definizioni successive. Oltre all'immagine da usare, viene eliminato il bordo, stabilite le dimensioni e il *padding*, cioè la distanza interna tra bordo e contenuto.

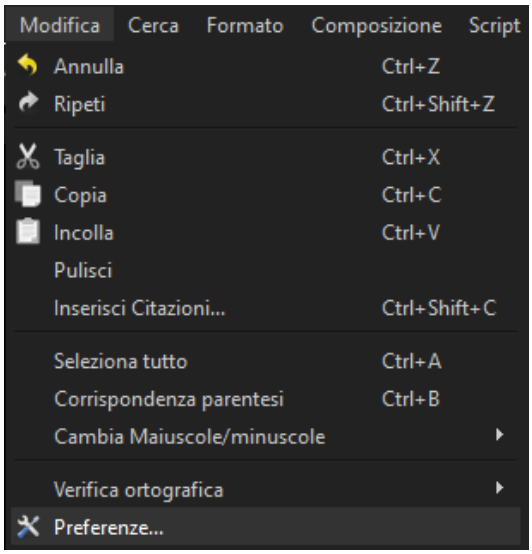

Figura 4. QMenu e QMenuBar.

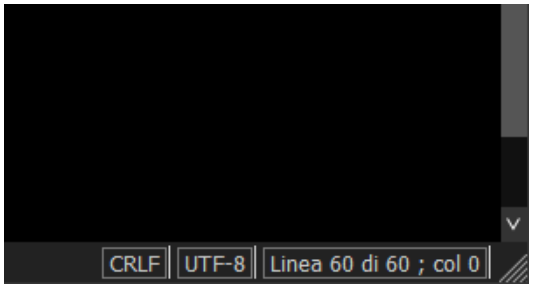

Figura 5. QScrollBar

L'ultima definizione stabilisce l'aspetto dello sfondo della barra sopra e sotto alla parte in movimento: con none è a tinta unita, senza questa indicazione apparirebbe a scacchi.

Definizione 6. QScrollBar:horizontal

```
QScrollBar:horizontal {
    background: #111111;
    height: 15px;
    margin: 0 20px 0 20px;
}
QScrollBar::handle:horizontal {
    background: #555555;
    min-width: 20px;
}
QScrollBar::add-line:horizontal {
     border: none;
     background: #333333;
    width: 20px;
    subcontrol-position: right;
    subcontrol-origin: margin;
}
QScrollBar::sub-line:horizontal {
     border: none;
     background: #333333;
    width: 20px;
    subcontrol-position: left;
    subcontrol-origin: margin;
}
QScrollBar::add-line:horizontal:pressed {
    background: #444444;
```

```
}
QScrollBar::sub-line:horizontal:pressed {
    background: #444444;
}
QScrollBar:left-arrow:horizontal {
    image: url(./res/leftarrow.png);
    border: none;
    width: 8px;
    height: 8px;
    padding: 12px;
}
QScrollBar::right-arrow:horizontal {
    image: url(./res/rightarrow.png);
    border: none;
    width: 8px;
    height: 8px;
    padding: 12px;
}
QScrollBar::add-page:horizontal, QScrollBar::sub-page:horizontal {
    background: none;
}
```

Le definizioni per le barre di scorrimento orizzontali sono replicate da quelle verticali con le opportune modifiche del caso.

Definizione 7. QSizeGrip

```
QSizeGrip {
    image: url(./res/sizegrip.png);
    background: none;
}
```

Questa semplice definizione è necessaria per poter visualizzare, nell'angolo in basso a destra, le righe diagonali che segnalano la zona da trascinare per ridimensionare la finestra (la zona è visibile nella figura 5).

Definizione 8. QTabWidget

```
QTabWidget::pane {
    border: 1px solid #333333;
}
QTabWidget::tab-bar {
    left: 5px;
}
QTabWidget QToolButton {
    background-color: #444444;
    margin: 2px;
    padding: 1px;
}
QTabWidget QToolButton:hover {
    background-color: #555555;
}
```

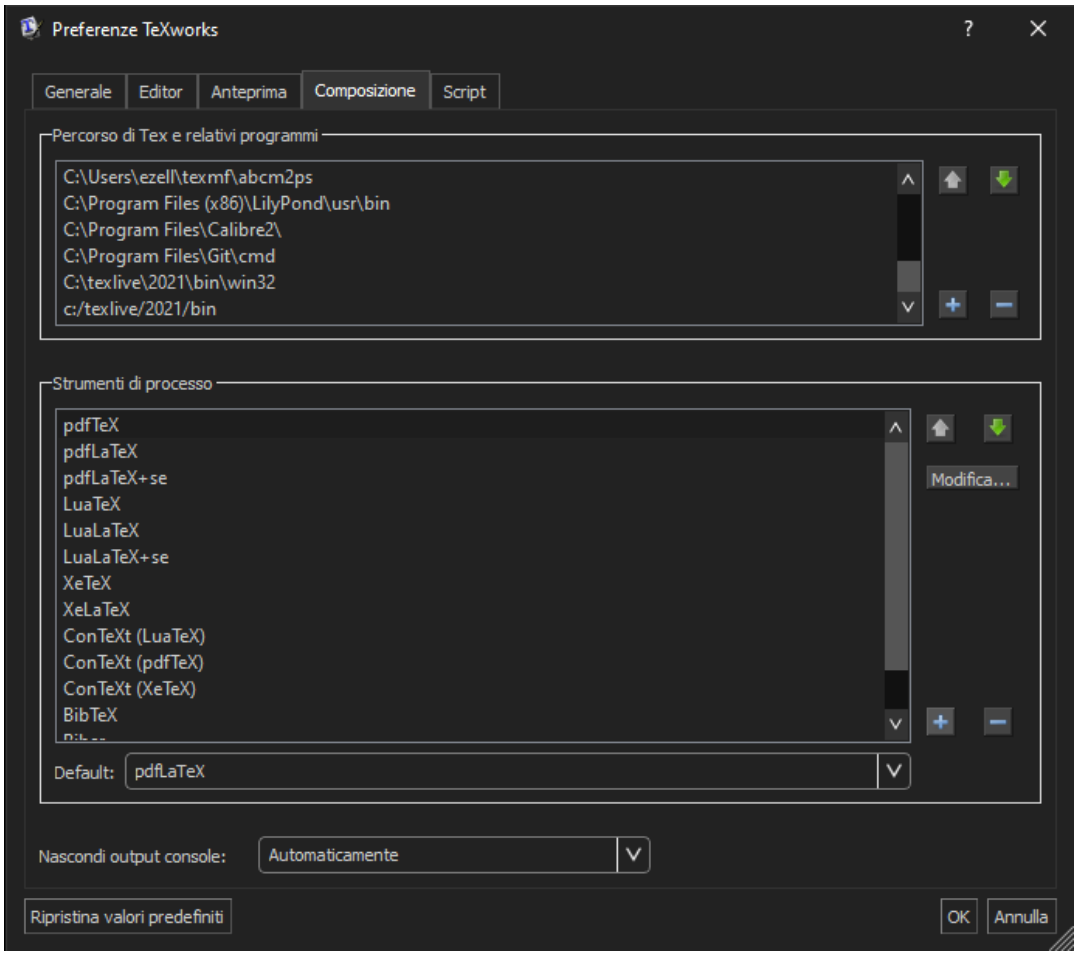

Figura 6. QTabWidget e QTabBar

`QTabWidget` è il riquadro che contiene le linguette che permettono di aprire schede diverse (come nella finestra delle preferenze, vedi la figura 6). È necessario inserire almeno la prima definizione per poter personalizzare anche `QTabBar`, l'elemento che controlla l'aspetto delle linguette vere e proprie. La seconda definizione sposta l'area delle linguette a sinistra di 5 pixel, soluzione che visivamente ho trovato preferibile.

Le ultime due definizioni impostano l'aspetto dei pulsanti all'interno di questa tipologia di riquadri; tra questi rientra anche il pulsante di chiusura della console, il quale, però, necessita di impostazioni diverse e verrà sistemato con la definizione 10.

Definizione 9. QTabBar

```
QTabBar::tab {
    background: #222222;
    border: 1px solid #555555;
    border-bottom-color: #333333;
    min-width: 8ex;
    padding: 4px 8px 4px 8px;
```

```
}
QTabBar::tab:selected {
    background: #333333;
    color: #ffffff;
    border-color: #555555;
    border-bottom-color: #333333;
}
QTabBar::tab:hover {
    background: #444444;
}
QTabBar::tab:!selected {
    margin-top: 2px;
}
```

Le prime due definizioni regolano l'aspetto delle linguette, rispettivamente nella versione normale e quando una linguetta è selezionata. Si noti come nella seconda ho modificato anche il colore del testo per farlo apparire più in risalto. Inoltre, in entrambe viene definito il colore del bordo, ma il colore del bordo inferiore è lo stesso impostato in `QTabWidget::pane`, ovvero il riquadro del contenuto della linguetta: in questo modo, la linguetta sembrerà un'estensione del riquadro (che è il comportamento che visivamente ci si aspetta). La terza definizione imposta anche un ulteriore colore di sfondo visibile al passaggio del mouse.

L'ultima definizione, assolutamente facoltativa, è un trucchetto carino: fa sì che tutte le linguette *non* selezionate siano più piccole di 2 pixel, mettendo così in risalto la linguetta selezionata.

Definizione 10. ClosableTabWidget

```
Tw--UI--ClosableTabWidget QToolButton {
    qproperty-icon: url(./res/close.png);
    background-color: #333333;
}
Tw--UI--ClosableTabWidget QToolButton:hover {
    background-color: #800000;
}
```

Queste due definizioni servono a modificare l'aspetto del pulsante di chiusura della console (figura 7). Appare subito evidente come il nome dell'elemento richiamato sia diverso da quelli visti finora. Questo perché tutti gli elementi modificati sono quelli standard previsti dal modello Qt, mentre `ClosableTabWidget` è un elemento creato dagli autori di TEXworks appositamente, quindi bisogna specificare il suo 'indirizzo' completo per poterlo richiamare (devo ringraziare uno degli autori, Stefan Löffler, che mi ha indicato il percorso corretto). Le definizioni vanno a modificare *tutti* i pulsanti di tipo `QToolButton` contenuti nell'elemento: al momento è uno solo, quello di chiusura, quindi alla versione 0.6.6 il codice è funzionante, ed è l'unico possibile. Dopo lo scambio con l'autore avvenuto su GitHub, posso anticipare che dalle prossime versioni il pulsante di chiusura avrà un nome proprio, un *id* (`closeButton`) che potrà essere richiamato e modificato direttamente, senza andare a toccare eventuali altri pulsanti che potrebbero venire aggiunti in futuro.

C'è un'altra differenza rispetto ai casi precedenti: per indicare un'immagine per il pulsante (anche in questo caso la croce originale è di colore scuro e non si vedrebbe con il nostro tema),

Figura 7. La console: il pulsante in rosso è ottenuto al passaggio del mouse

la proprietà `image` usata finora non funziona. Questo perché Qt non accetta da un foglio di stile esterno qualsiasi tipo di modifica a qualsiasi elemento, ma per ciascun elemento si possono modificare solo alcune proprietà prestabilite (si veda la documentazione riportata a inizio sezione). La soluzione è stata usare il comando `qproperty` che permette di modificare direttamente le proprietà di un elemento, tra cui l'immagine, e funziona anche dove non è previsto l'uso di `image`.

La seconda definizione imposta un colore di sfondo rosso scuro al passaggio del mouse.

Definizione 11. QDockWidget

```
QDockWidget {
    border: 1px solid lightgray;
    titlebar-close-icon: url(./res/close.png);
    titlebar-normal-icon: url(./res/undock.png);
}
QDockWidget::title {
    text-align: left;
    background: #222222;
    padding-left: 5px;
}
QDockWidget::close-button, QDockWidget::float-button {
    border: 1px solid transparent;
    background: none;
    padding: 0px;
}
QDockWidget::float-button:hover {
    background: #444444;
}
QDockWidget::close-button:hover {
    background: #800000;
}
```

`QDockWidget` è l'elemento che regola la finestra che contiene i risultati di una ricerca, o, per essere più specifici, la cornice della finestra con la barra del titolo (area in verde nella figura 8). La prima definizione imposta un bordo e le icone per i due pulsanti, quello di chiusura e quello per disancorare la finestra e renderla flottante (in azzurro nella figura 8). La seconda

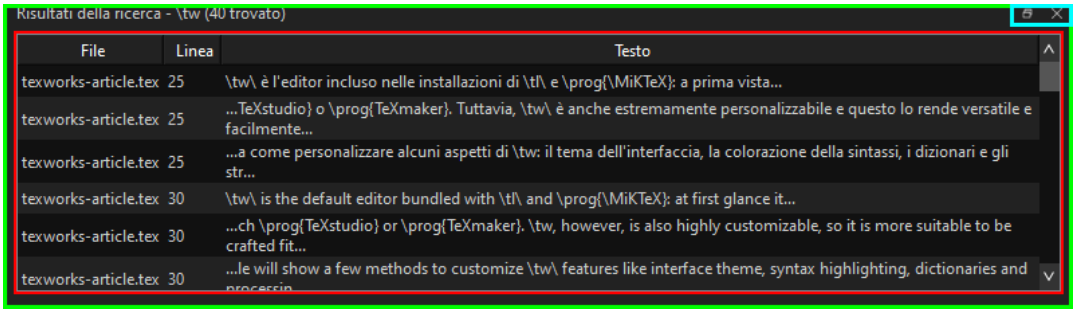

Figura 8. QDockWidget e la tabella definita da QHeaderView e QTableView

definizione regola l'aspetto della barra del titolo; la terza l'aspetto dei due pulsanti nel loro stato normale. La definizione raggruppa assieme due elementi: per Qt la sintassi è valida, l'importante è che a noi stia bene dare a due oggetti distinti le medesime caratteristiche. Trattandosi di due pulsanti, è corretto uniformarne l'aspetto; con le due seguenti definizioni invece ne differenzio il colore di sfondo al passaggio del mouse (preferisco sempre colorare di rosso i pulsanti di chiusura per evitare pressioni accidentali).

Definizione 12. QHeaderView & QTableView

```
QHeaderView::section  {
    background-color: #222222;
    border-right: 1px solid lightgray;
    border-left: 1px solid lightgray;
    color: white;
    padding-left: 4px;
}
QHeaderView::section:hover  {
    background-color: #444444;
}

QTableView {
   background-color: #111111;
   alternate-background-color: #222222;
   selection-background-color: #555555;
   font-color: #FFFFFF;
}
```

Ho raggruppato questi due elementi perché assieme definiscono l'aspetto dei risultati della ricerca, che sono inseriti in una tabella (l'area in rosso nella figura 8). `QHeaderView` stabilisce l'aspetto della riga di intestazione, mentre `QTableView` l'aspetto delle righe sottostanti. Qui è stata usata anche la proprietà `alternate-background-color` per indicare un colore diverso per le righe pari, così da creare un'alternanza tra i colori delle righe e migliorare la visibilità.

Definizione 13. QToolButton

```
QToolButton:hover {
    background-color: #444444;
}
```

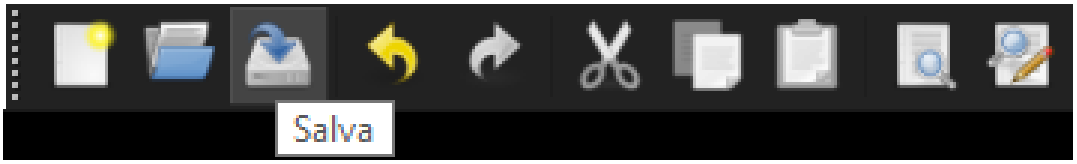

Figura 9. QToolButton

Con questa definizione si imposta un colore che si attiva al passaggio del mouse per tutti i pulsanti con immagini, come quelli immediatamente sotto alle voci di menu principale (visibili nella figura 9).

Definizione 14. QPushButton

```
QPushButton {
    background-color: #222222;
    border-style: outset;
    border-width: 1px;
    border-color: #555555;
    padding: 4px;
}
QPushButton:pressed {
    background-color: #555555;
    border-style: inset;
    border-width: 1px;
    border-color: #555555;
    padding: 4px;
}
QPushButton:hover {
    background-color: #444444;
}
```

`QPushButton` invece definisce i pulsanti con contenuto testuale (figura 10). In questo contesto è stato definito lo stile del bordo, per rendere una sensazione di tridimensionalità dei pulsanti.

Definizione 15. ScreenCalibrationWidget

```
Tw--UI--ScreenCalibrationWidget {
    color: #000000;
}
```

Quest'ultima definizione si rivolge a un altro elemento specifico di TEXworks: il righello per la calibrazione dello schermo presente nelle preferenze (scheda 'anteprima'); come nel caso precedente, va indicato il percorso completo. Con la versione `0.6.6` di TEXworks il colore di sfondo del righello è bianco *hardcoded*, vale a dire che è definito direttamente nel codice del programma, e non si può modificare. Con questa definizione vado quindi a modificare in nero il colore di primo piano, che viene usato per disegnare le tacche e i numeri del righello. Ringraziando gli autori che hanno accolto il mio suggerimento, dalla prossima versione di TEXworks, anche il colore di sfondo del righello sarà modificabile e sarà quindi possibile renderlo coerente con il resto del tema.

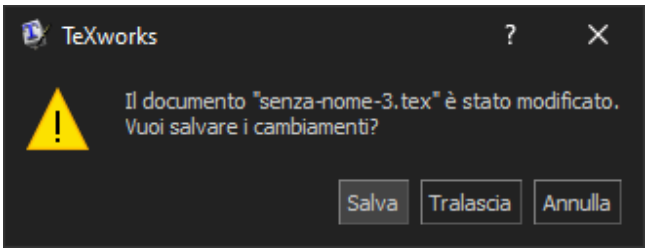

Figura 10. QPushButton

5.3. Suggerimenti

Tutte le definizioni viste finora permettono di ottenere un tema scuro funzionante; inoltre, le spiegazioni fornite a corredo dovrebbero essere sufficienti per consentire a chiunque di cimentarsi nel creare un tema colorato a piacimento. Prima di concludere, vorrei condividere alcuni suggerimenti frutto di molti esperimenti (ed errori!), che possono migliorare sensibilmente l'aspetto di un tema.

Il primo è la raccomandazione di usare il più possibile gli stati delle proprietà (*selected*, *pressed*, *hover*): in questo modo l'interfaccia apparirà dinamica, come se rispondesse alle attività dell'utente. Le prime versioni che ho creato non li prevedevano, e sembrava di avere tra le mani un programma vecchissimo e poco funzionante.

Il secondo consiglio riguarda la scelta della combinazione di colori. Chiaramente, andando a creare un tema ad uso personale, abbiamo la massima libertà di scelta. Tuttavia, credo sia comunque utile tenere in considerazione il rapporto di contrasto tra i colori desiderati, così da ottenere una maggiore leggibilità. Io mi sono rifatto alle raccomandazioni delle "Linee guida per l'accessibilità dei contenuti Web", le WCAG 2.1 (*Web Content Accessibility Guidelines*), dove, al criterio di successo 1.4.6 raccomandano un rapporto di contrasto di almeno 7 : 1. Esistono svariati strumenti online che permettono di calcolare il rapporto, io ho usato il seguente:

`https://juicystudio.com/services/luminositycontrastratio.php`

Un altro sito che uso spesso è `paletton.com`, uno strumento che permette, a partire dal codice `rgb` di un colore, di individuare, ad esempio, le gradazioni corrette da usare quando serve una variante più chiara o più scura (come per i cambi di stato di un elemento), oppure triadi (o anche tetradi) di colori abbinati tra loro secondo una serie di parametri configurabili (angolo di distanza nella ruota del colore, tonalità, saturazione, luminosità, contrasto).

6. La colorazione della sintassi

Una delle funzioni più utili di un editor di testo è la capacità di colorare la sintassi del linguaggio usato in un documento. In TEXworks abbiamo la possibilità di modificare le regole per la selezione della sintassi tramite delle *regular expression* (regex), di stabilire colori e formattazione personalizzati, e, volendo, anche di introdurre interi nuovi linguaggi. È utile approfondire la questione per almeno due motivi: da un lato la possibilità di modificare le regole ci permette di scegliere quale codice evidenziare, così da adattare il livello di dettaglio secondo le nostre preferenze; dall'altro la possibilità di modificare i colori è praticamente indispensabile se abbiamo

personalizzato l'interfaccia, perché tutte le regole predefinite usano tonalità pensate per un tema chiaro. In un'interfaccia con uno sfondo scuro i caratteri evidenziati risultano quasi invisibili.

Le regole che stabiliscono la colorazione sono salvate nel file `syntax-patterns.txt`, che si trova nella cartella `configuration`, la stessa che abbiamo già incontrato nella sezione 4. Il file che viene fornito assieme a TeXworks contiene già delle regole di base, suddivise in quattro linguaggi, chiamati: `LaTeX`, `LaTeX DTX`, `ConTeXt` e `BibTeX`. Qui prenderemo in esame solo il primo, ma una volta capito il meccanismo non sarà troppo difficile adattare le regole anche per altri tipi di file (cfr. nota 6). Il linguaggio predefinito `LaTeX` contiene cinque regole, che permettono di colorare: caratteri speciali, ambienti, pacchetti richiamati con `\usepackage`, comandi e commenti.

Nella prima parte del file è inoltre presente una sezione commentata che riporta già tutte le indicazioni di sintassi per inserire una regola. La struttura di base è la seguente:

⟨*stile testo*⟩ ⟨*controllo ortografico*⟩ ⟨*regex*⟩

Lo stile del testo prevede fino a tre tipi di definizioni:

1. Il colore del testo
2. Lo sfondo del testo
3. Un'indicazione della forma (*fontflag*):

   B grassetto (*bold*)
   I corsivo (*italic*)
   U sottolineato (*underlined*)

Il colore può essere indicato o con il nome secondo lo standard SVG, o con il valore `rgb`, nel formato esadecimale `#rrggbb` (in questo caso, va inserito uno spazio vuoto a inizio riga per evitare che la riga venga considerata un commento). La sintassi da usare è la seguente:

⟨*colore del testo*⟩ / ⟨*colore sfondo*⟩ ; ⟨*forma*⟩

La barra separa il colore del testo da quello di sfondo, il punto e virgola precede la forma del carattere.

Il controllo ortografico indica se per quella sezione di testo deve essere attivato il controllo ortografico, e va indicato solamente con `Y`(es) o `N`(o). In genere, sarà 'no' per tutte le definizioni, tranne che per quella dei commenti, solitamente scritti nella stessa lingua del documento.

L'ultima voce è la *regular expression*, la funzione che ha il compito di individuare una specifica stringa di testo nel documento: sarà proprio configurando tante *regex* che potremo assegnare colori e formattazioni ai vari elementi della sintassi LaTeX.

Prima di procedere con alcuni esempi, è bene ricordare che esiste una sorta di gerarchia tra le regole, stabilita dal loro ordine, per cui una regola più specifica va sempre inserita prima di una più generale.

Il primo esempio è la riga di definizione all'inizio della sezione del linguaggio `[LaTeX]`, la quale specifica di colorare di arancione i caratteri speciali.

```
# special characters
orange      N       [$#^_{}&]
```

In questa definizione sono chiaramente visibili le tre parti: colore, controllo ortografico e *regular expression*. Tra le parentesi quadre sono racchiusi tutti i caratteri da individuare.

Applicando la stessa logica, possiamo provare a individuare anche altri elementi. Consiglio di lasciare inalterati i linguaggi predefiniti, e crearne uno personalizzato dove potremo eseguire tutte le prove che vogliamo, senza correre il rischio di danneggiare qualcosa. Il modo più rapido è quello di creare una copia di tutto il linguaggio `[LaTeX]` con le sue definizioni e rinominare la copia come preferiamo, ad esempio `[LaTeX_test]`.

Qui di seguito elencherò alcune definizioni da aggiungere a questo linguaggio, assieme a un commento su come ho costruito la stringa *regex*.

```
# LaTeX refs
darkseagreen    N   \\(?:ref|eqref|pageref)\s*\{[^}]*\}

# LaTeX labels
lightseagreen   N   \\(?:label)\s*\{[^}]*\}
```

Queste due definizioni colorano i comandi per ottenere i riferimenti incrociati: ho usato due sfumature diverse dello stesso colore, il verde, per distinguere il comando `\label` che genera le etichette, dagli altri che le richiamano. La stringa di ricerca è stata ottenuta in questo modo:

```
\\
% trova il carattere \.
% analogamente a LaTeX, per indicare in maniera letterale i caratteri speciali che svolgono una funzione
    particolare in una regular expression bisogna farli precedere dal backslash.

(?:ref|eqref|pageref)
% gruppo che trova alternativamente uno dei tre comandi indicati. Il segno | funziona come l'operatore
    booleano OR. Questo è un gruppo 'non-capturing' ovvero che trova senza catturare, senza
    assegnare un ID.

\s*
% trova uno spazio (comprese tabulazioni e fine riga). Il carattere * è un quantificatore: recupera la
    ricerca che lo precede da 0 a infinite volte.

\{
% trova il carattere {.

[^}]*
% trova un qualsiasi carattere che non sia } da 0 a infinite volte.

\}
% trova il carattere }.
```

La stringa per le etichette è composta in maniera analoga.

Per chi si volesse cimentare nella creazione di stringhe personalizzate, consiglio il sito:

`https://regex101.com/`

La schermata principale presenta un riquadro in alto che permette di costruire una stringa di ricerca, un riquadro sottostante dove è possibile inserire del testo per verificare il funzionamento della stringa e una barra laterale a destra che contiene in alto uno strumento di analisi della stringa costruita e in basso un sintetico riepilogo dei comandi e dei *token* disponibili per costruire le stringhe. È uno strumento davvero utile che permette di risparmiare tantissimo tempo.

Vediamo ora la stringa per evidenziare i comandi di sezionamento.

```
# LaTeX sections
limegreen   N   \\(?:chapter|section|subsection|subsubsection)\s
    *(?:\[[^]]*\]\s*)?\{[^}]*\}
```

Come nell'esempio precedente, analizziamo la struttura della stringa.

```
\\
% trova il carattere \.

(?:chapter|section|subsection|subsubsection)
% gruppo che trova una tra le parole proposte.

\s*
% torna in quasi tutte le definizioni, poiché permette uno spazio, che comprende anche il fine riga.

(?:\[[^]]*\]\s*)?
% altro gruppo che serve a trovare un eventuale argomento opzionale del comando di sezionamento:
    prima trova il carattere [ poi trova un numero indefinito di caratteri che non siano ], poi il
    carattere ], anche in questo caso viene ammesso lo spazio, e infine c'è il quantificatore ? che
    ritrova l'argomento che lo precede 0 o 1 volta.

\{[^}]*\}
% analogamente, questa sequenza trova l'argomento obbligatorio: carattere {, numero indefinito di
    caratteri che non siano } e infine } che chiude il gruppo.
```

Quella che segue è la soluzione che ho trovato per i comandi di citazione; dovrebbe supportare sia la maggior parte dei comandi di biblatex sia quelli dell'accoppiata BibTeX–natbib.

```
# LaTeX cite
lightcoral  N   \\(?i)(paren|text|foot|smart|super|auto)?cite([pt
    ]|author|year|date|url|title)?[*]?\s*(\[[^]]*\]){0,2}\{[^}]*\}
```

La stringa è piuttosto lunga, proviamo a vederla nel dettaglio.

```
\\
% backslash di inizio comando.

(?i)
% questo non è un gruppo, ma un flag: indica di recuperare tutta la stringa che segue indipendentemente
    che sia scritta in maiuscolo o minuscolo (*case (I)nsensitive*). Questo perché biblatex prevede per i
    suoi comandi una variante con la maiuscola.

(paren|text|foot|smart|super|auto)?
% biblatex usa il comando \cite e dei suoi composti, questo gruppo può trovare il prefisso del
    composto, il quantificatore ? trova 0 o 1 occorrenza.

cite([pt]|author|year|date|url|title)?
```

```
% qui viene chiesto direttamente di trovare la parola cite, con un'eventuale suffisso, inserito nel gruppo
    tra parentesi: p, t, author e year sono usati da natbib, questi ultimi e i rimanenti anche da
    biblatex.[8]

[*]?
% Viene ricercato letteralmente il carattere asterisco, serve perché sia natbib sia biblatex prevedono
    delle varianti asteriscate di alcuni comandi.

\s*
% solito comando per gli spazi.

(\[[^]]*\]){0,2}
% un comando di citazione di biblatex può avere 0,1 o 2 argomenti opzionali tra quadre. La stringa qui
    sopra è composta quindi con un gruppo che cattura tutto ciò che è contenuto tra due quadre, e da
    un quantificatore tra graffe che indica il numero di occorrenze: da zero a due.

\{[^}]*\}
% anche qui viene ricercato il contenuto tra due parentesi graffe, che costituiscono l'argomento
    obbligatorio con la chiave del comando di citazione.
```

Concludo questa sezione di definizioni con una soluzione per gli ambienti matematici in linea, ricavata da una risposta su Stack Overflow (2013); chi volesse approfondire l'argomento è invitato a leggere la risposta originale sul sito.

```
# math inline
hotpink N   (?<!\\)((?<!\$)\$(?!\$))(.*?)(?<!\\)(?<!\$)\1(?!\$)
```

Va sottolineato, tuttavia, che la stringa al momento non funziona, perché il motore regex di TEXworks per ora non supporta la funzione *lookbehind*. TEXworks è costruito con il toolkit Qt, che ha un suo motore interno per le espressioni regolari: quello attuale si chiama QRegularExpression, che ha sostituito il precedente QRegExp. Sebbene il nuovo costituisca un notevole miglioramento del precedente, non è ancora un motore "PCRE" (*Perl-compatible regular expression*), e alcune espressioni continuano a non essere disponibili in TEXworks. Se la situazione in futuro dovesse cambiare, la stringa appena vista potrebbe diventare molto utile.

Nel frattempo, per evidenziare la matematica in linea, ho optato per la soluzione molto più banale di evidenziare di un colore sgargiante il simbolo del dollaro, così da individuarlo immediatamente in mezzo al testo.[9]

```
# special characters
hotpink     N   \$
```

Come potete notare, tutte le definizioni viste finora non includono mai un cambio del colore di sfondo perché personalmente lo trovo una distrazione, tuttavia è stato indicato come fare, e ognuno si senta libero di sperimentare.

8. Per il comando `\cites` usato da biblatex ho creato un secondo comando a parte, poiché non sono riuscito a individuare un modo efficace di combinare i due. Ammetto la mia scarsa conoscenza di regex quindi non escludo sia possibile integrarli.
9. Se volete usufruire di questa soluzione, dovete ricordarvi di eliminare il segno del dollaro dalla prima definizione vista in questa sezione, quella che permette di colorare i caratteri speciali.

Riferimenti bibliografici

Delmotte, A., Löffler, S. *et al.* (2021). *A short manual for TEXworks*. URL https://github.com/TeXworks/manual/releases/download/2021-03-08/TeXworks-manual-en.pdf.

Gregorio, E. (2010). «Introduzione a TEXworks». URL http://profs.sci.univr.it/~gregorio/introtexworks.pdf.

Stack Exchange (2017). «How can i set a dark theme in texworks?» https://tex.stackexchange.com/questions/383271/how-can-i-set-a-dark-theme-in-texworks. [Online; ultimo accesso 22 agosto 2021].

Stack Overflow (2013). «Regex to match latex equations». https://stackoverflow.com/questions/14182879/regex-to-match-latex-equations/14537848#14537848. [Online; ultimo accesso 22 agosto 2021].

WCAG 2.1 (2018). «Web content accessibility guidelines (WCAG) 2.1 (traduzione italiana)». https://www.w3.org/Translations/WCAG21-it/. [Online; ultimo accesso 22 agosto 2021].

Filippo Vomiero
Università degli studi di Padova
filippo.vomiero@unipd.it